МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт Информационных Технологий

Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА

по дисциплине

«Системный анализ данных СППР»

Студент группы:ИКБО-42-23 Голев С.С.
(Ф. И.О.студента)

Преподаватель Железняк Л.М.
(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Муравьиный алгоритм относится к роевым методам оптимизации, основанным на моделировании поведения колоний муравьёв при поиске кратчайших путей между источником пищи и гнездом. Основная идея заключается в использовании механизма обмена информацией через феромоны — следы, оставляемые муравьями на маршруте. Эти следы постепенно усиливаются при прохождении по более выгодным путям и испаряются со временем, что обеспечивает баланс между исследованием новых направлений и закреплением найденных решений. В работе рассматривается применение муравьиного алгоритма для решения задачи коммивояжёра. Такой подход позволяет эффективно находить приближенные оптимальные решения в сложных комбинаторных пространствах.

4 МУРАВЬИНЫЙ АЛГОРИТМ

Алгоритм основан на поведении муравьев, и представляет из себя стаю муравьёв перемещающихся на графе для поиска наилучшего пути.

4.1 Цель и задачи практической работы

Целью практической работы является изучение принципов муравьиного алгоритма и его применение для решения задач нахождения кратчайшего пути. В рамках работы необходимо освоить механику работы феромонов, которые помогают найти лучший путь

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Выполнить ручной расчёт одной итерации муравьиного алгоритма для гамильтонова графа;
2. Реализовать муравьиный алгоритм на языке Python для автоматического поиска кратчайшего пути;
3. Применить алгоритм для решения задачи коммивояжёра;
4. Сравнить эффективность результатов ручного расчёта и программной реализации, выявив преимущества алгоритма для задач оптимизации.

4.2 Постановка задачи

В рамках практической работы необходимо реализовать роевый алгоритм вручную и кодово, алгоритм будет проверятся на задаче коммивояжёра где граф состоит из шести вершин.

4.3 Ручной расчёт

Выполним ручной расчёт одной итерации, для гамильтонова графа.

Сначала опишем формулы, которые будут использоваться при расчёте.

Перемещение по графу основывается на вероятности, с которой выбирается следующая вершина и рассчитывается по формуле:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha} * \mu_{ij}^{\beta}}{\sum_{j \in N_i^k} \tau_{ij}^{\alpha} * \mu_{ij}^{\beta}}, \text{если } j \in N_i^k \\ 0, \text{в ином случае} \end{cases} \quad (4.1)$$

Здесь N_i^k представляет множество возможных вершин, связанных с -й вершиной, для k-го муравья. Если для любого i-го узла и k-го муравья $N_i^k = \emptyset$, тогда предшественник узла i включается в N_i^k. В этом случае в пути возможны петли. Когда все муравьи построили полный путь от начальной до конечной вершины, удаляются петли в путях, и каждый муравей помечает свой построенный путь, откладывая для каждой дуги феромон в соответствии со следующей формулой.

$$\Delta\tau_{ij}^{k}(t) = \frac{q}{L^k(t)} \quad (4.2)$$

Здесь $L^k(t)$ – длина пути, построенного k-м муравьем в момент времени t.

Таким образом, для каждой дуги графа концентрация феромона определяется следующим образом:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^{k}(t) \quad (4.3)$$

где n_k - число муравьев.

Построим матрицу, которая будет описывать граф для ручного расчёта.

Таблица 4.1 – Значения матрицы

	1	2	3	4	5
1		10	15	20	25
2	10		35	25	30
3	15	35		30	10
4	20	25	30		15
5	25	30	10	15	

Исходное количество муравьев: 3.

Начальные феромоны для всех ребер: 0.2.

Начальная вершина: 1.

Рассчитаем вероятность перехода из первой вершины в каждую Вершину, для этого используем формулу 4.1:

$P_{10} \approx 0.002$

$P_{12} \approx 0.06035$

$P_{13} \approx 0.1183$

$P_{14} \approx 0.0820$

Сделаем выбор для муравьев:

Муравей 1: переходит в 0

Муравей 2: переходит в 3

Муравей 3: переходит в 4

Далее рассмотрим полный путь только одного муравья, текущий путь которого равен $1 \rightarrow 0$.

По формуле 4.1 рассчитаем вероятность для следующего перехода:

$P_{02} \approx 0.5200$

$P_{03} \approx 0.2926$

$P_{04} \approx 0.1874$

Переходим в 3-ю вершину, рассчитаем вероятности для следующих вершин, на данный момент путь муравья 1 →0 →3.

P_{32}=0.2

P_{34}=0.8

Переходим в 4-ую вершину. Далее переходим во вторую, так как она осталась последней, и замыкаем цикл, в итоге первый муравей имеет путь 1 →0 →3 →4 →2 →1 длинна которого 90.

По формуле 4.2 получаем что для каждого ребра, который прошёл первый муравей добавляется 1.112 феромоны, добавление идёт после того как все муравьи пройдут свой путь и рассчитается по формуле 4.3.

Проходы по ребрам для остальных муравьев остаётся идентичным, поэтому не рассматриваются.

При следующих итерациях данные шаги будут повторяться и муравьи будут находить наилучшее решение задачи Коммивояжёра.

4.4 Результат работы

Реализуем нахождение кратчайшего пути с помощью муравьиного алгоритма, количество муравьев: 10, количество итераций: 100, выполним реализацию на языке Python. Реализация представлена в приложении Г.

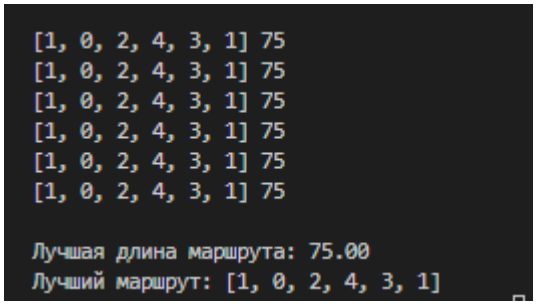

Рисунок 4.1 – Пример выполнения муравьиного алгоритма

4.5 Результат работы нахождения кратчайшего пути

В ходе практической работы был выполнен ручной расчёт одной итерации муравьиного алгоритма.

Данный расчёт демонстрирует работу муравьиного алгоритма: муравьи находят кратчайший путь, основываясь на ферамонах, которые оставляют другие муравьи прошедшие эти пути.

Далее была выполнена кодовая реализация данного алгоритма, которая позволила:

1. Автоматически находить кратчайший путь;
2. Применять алгоритм для решения задачи коммивояжёра;

В результате работы показано, что муравьиный алгоритм является эффективным методом для нахождения кратчайшего пути и может использоваться в многих практических задачах использующих графы.

ЗАКЛЮЧЕНИЕ

В ходе работы был реализован муравьиный алгоритм для решения задач комбинаторной оптимизации. Эксперименты показали, что данный метод позволяет эффективно находить приближённые оптимальные решения, минимизируя длину маршрута в задаче коммивояжёра. За счёт механизма обновления феромонов алгоритм адаптируется к найденным удачным решениям и постепенно усиливает перспективные направления поиска. Коллективное поведение агентов обеспечивает баланс между исследованием новых путей и использованием накопленного опыта. Полученные результаты подтверждают эффективность и устойчивость муравьиного алгоритма, а также его пригодность для решения широкого круга оптимизационных задач.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 15.09.2025).

2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.

3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.

4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

ПРИЛОЖЕНИЯ

Приложение Г – Код программы “Муравьиный алгоритм”

Приложение Г

Код программы Муравьиный алгоритм

Листинг Г.1 — Основной алгоритм программы

```
import numpy as np
import random

def createGraph(cNodes):
    structure = {}
    step = 1
    for i in range (cNodes):
        structure[str(i + step)] = []

    for i in range (cNodes):
        for j in range (i + 1,cNodes):
            w = np.random.randint(1, 6)

            structure[str(i + step)].append((str(j + step), w))
            structure[str(j + step)].append((str(i + step), w))

    return structure, step

def aco_tsp(graph, startNode, num_ants=10, iterations=100,
alpha=1):
    nodes = list(graph.keys())
    n = len(nodes)

    tau = {i: {j: np.random.uniform(0.1, 1.0) for j, _ in graph[i]}
for i in graph}
    eta = {i: {j: 1 / w for j, w in graph[i]} for i in graph}

    best_length = float('inf')
    best_path = None

    for t in range(iterations):
        paths = []
        lengths = []

        for _ in range(num_ants):
            start = str(startNode)
            unvisited = set(nodes)
            unvisited.remove(start)
            path = [start]
            total_dist = 0
            current = start

            while unvisited:
                neighbors = [n for n, _ in graph[current] if n in
unvisited]
                if not neighbors:
                    break
```

Листинг Г.2 — Продолжение листинга Г.1

```
                probs = []
                denom = sum((tau[current][j] ** alpha) for j in
neighbors)
                for j in neighbors:
                    p = (tau[current][j] ** alpha) / denom
                    probs.append(p)

                next_node = random.choices(neighbors,
weights=probs, k=1)[0]
                dist = next(w for wj, w in graph[current] if wj ==
next_node)
                total_dist += dist
                path.append(next_node)
                unvisited.remove(next_node)
                current = next_node


            if len(path) == n:
                last, first = path[-1], path[0]
                dist = next(w for wj, w in graph[last] if wj ==
first)
                total_dist += dist
                path.append(first)

            # print(' -> '.join(path))
            # print(total_dist)

            paths.append(path)
            lengths.append(total_dist)

            if total_dist < best_length:
                best_length = total_dist
                best_path = path


        for k in range(num_ants):
            path = paths[k]
            length = lengths[k]
            for i in range(len(path) - 1):
                a, b = path[i], path[i + 1]
                tau[a][b] += 1 / length
                tau[b][a] += 1 / length

    return best_path, best_length
```

Листинг Г.3— Продолжение листинга Г.2

```
if __name__ == '__main__':
    cNodes = 6
    structure, start = createGraph(cNodes)

    path, weight = aco_tsp(structure, startNode=start, num_ants =
cNodes)
    print("Кратчайший путь:", ' -> '.join(path))
    print("Длина:", weight)
```